ARMY RESEARCH LABORATORY

Design and Analysis of a Parallel, Real-Time, Automatic Target Recognition Algorithm

by Philip David, Philip Emmerman,
and Sean Ho

ARL-TR-1112

September 1996

19961011 038

DTIC QUALITY INSPECTED 1

REPORT DOCUMENTATION PAGE

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 1996	3. REPORT TYPE AND DATES COVERED Final, from March 1993 to March 1994

4. TITLE AND SUBTITLE

Design and Analysis of a Parallel, Real-Time, Automatic Target Recognition Algorithm

5. FUNDING NUMBERS

PE: 62120A

6. AUTHOR(S)

Philip David, Philip Emmerman, and Sean Ho

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

U.S. Army Research Laboratory
Attn: AMSRL-IS-SA
2800 Powder Mill Road
Adelphi, MD 20783-1197

8. PERFORMING ORGANIZATION REPORT NUMBER

ARL-TR-1112

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Army Artificial Intelligence Center
Attn: SAIS-AI
107 Army Pentagon
Washington D.C. 20310-0107

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

AMS code: 612120.H1600
ARL PR: 360P61

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Automatic target recognition (ATR) is made difficult by the variety of conditions under which an ATR system may be required to operate. Because the number of operations required to execute a particular ATR algorithm can vary greatly from one scenario to another, a fixed hardware and software architecture usually will not be able to execute a given ATR algorithm in all required scenarios within some given real-time constraints. A solution to this problem is to use a scalable architecture. The hardware and software of such an architecture can easily be scaled to meet the processing requirements of a particular scenario. This report describes a scalable architecture system that we developed that implements a real-time ATR algorithm.

14. SUBJECT TERMS

ATR, scalable architecture, parallel processing

15. NUMBER OF PAGES 18

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Contents

Figures

Tables

3

1. Introduction

Automatic target recognition (ATR) is the process of locating and recognizing targets in data generated by one or more sensors. ATR is made difficult by the variety of conditions under which an ATR system may be required to operate (e.g., targets may be occluded; targets may have low target-to-background contrast; there may be a necessity to recognize a variety of targets; target appearance can vary greatly with different viewpoints; natural and manmade clutter may be present in the scene, etc). Therefore, computation required to execute a particular ATR algorithm can vary greatly from one scenario to another, and a fixed hardware and software architecture will not usually be able to execute a given ATR algorithm in all required scenarios within some given real-time constraints.

A solution to this target-recognition problem is to use a processing architecture that can easily be scaled to meet the processing requirements of a particular scenario. A scalable architecture is a computer architecture that can deliver an increase in performance proportional to an increase in its size; however, efficiently using such an architecture requires a software architecture that scales along with the hardware.

For our work, we employed the ATR Relational Template Matching (ARTM) algorithm,* which uses a hierarchy of target silhouette models to detect and recognize targets in infrared (IR) imagery. We converted the original, sequential algorithm into a parallel, scalable algorithm that runs on a scalable architecture consisting of Texas Instruments TMS320C40 processors. In this report, we describe how we decomposed, distributed, and ran the ATR algorithm on the C40s, using their parallel, high-speed, interprocessor communication links to approach maximum system performance.

The report is organized as follows. First, we describe the system's hardware architecture and its operating system. We then present a description of the ARTM algorithm and its parallel implementation, followed by an analysis of how well the algorithms scaled when they were applied to more difficult problems. We conclude with a discussion of the system's performance.

2. Scalable Architectures

Recent advances in hardware and software make the development of low-cost, scalable architectures more practical. At the digital signal processor (DSP) chip level, vendors are adding more powerful input/output (I/O) and interconnect features that allow designers to combine multiple DSPs more efficiently and with greater flexibility. At the board level, vendors are packaging multiple DSPs on single boards, using a broad range of point-to-point and shared-memory configurations.

The scalable system that runs the automatic target recognition (ATR) code is a heterogeneous system that consists of a general-purpose, single-board computer (SBC) and a DSP subsystem. Both are VME (VERSAmodule Europe) bus-based, commercial off-the-shelf (COTS) computer boards (see fig. 1). The SBC uses a 25-MHz Motorola 68040 processor. This board services requests (e.g., I/O to a remote file system) from the DSP subsystem. During software development and testing, binary codes and data are forwarded from the Unix-based development environment to the DSP via this SBC.

The second board in the system is a quad-C40 board that consists of four 40-MHz TI TMS320C40 DSP central processing units (CPUs). Each DSP CPU is capable of performing 275 million operations per second (MOPS), and has a maximum data throughput of 320 Mb/s, including 20 Mb/s throughput from each of its six interconnected communications ports (see fig. 2). Three of six communication ports on each C40 can be externally connected to other C40s. In doing so, scalability can be achieved by adding C40 boards to the system as needs change. Another important feature of the C40 is its use of direct memory access (DMA), which permits data to be transferred between memories without the intervention of the DSP's CPU [1,2]. The quad-C40 board has 2 Mb of global static random access memory (SRAM) and 2 Mb of SRAM for each of the four processors.

Figure 1. System overview.*
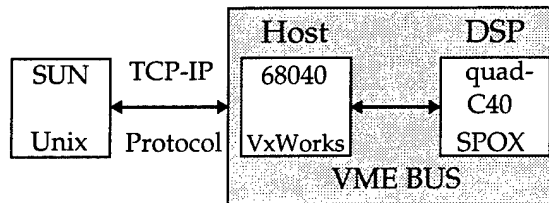
* *Three operating systems are involved in running the ATR code on the C40s.*

Figure 2. The overall architecture of the quad-C40 DSP board.*
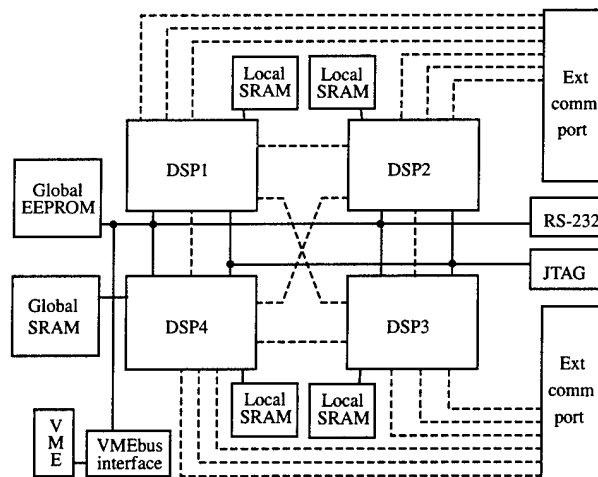
* *Dashed lines represent the parallel communication ports.*

3. Operating System

Since the host SBC (68040) directs and synchronizes all tasks run on the DSP board and other boards on the VME bus, vxWorks,* a real-time multi-tasking operating system, is run on the SBC. In order to minimize the time that is spent in debugging and porting the ATR code from the Unix development environment to the DSP platform and to preserve as much of the original code as possible, we required a Unix-compatible DSP operating system (OS). It was necessary that the OS provide the basic features of a real-time operating system, in addition to making the application easily portable to other DSP systems in the future without modifying the already-developed ATR code. We selected SPOX,† which also provides such features as dynamic memory allocation from multiple memory segments and a C standard I/O library. The C standard I/O server allows the host server to communicate with SPOX tasks running on the DSP board with C standard library functions such as fopen(), printf(), and scanf() [3].

4. ATR Application

4.1 ARTM Algorithm

The ARTM algorithm is a model-based, target-recognition algorithm that consists of an offline algorithm design process and an online target-recognition process [4,5]. The offline process uses CAD models to construct a decision tree of templates that are matched to imagery during the online process. Each node in the decision tree represents a test for the presence of a target silhouette boundary. The tree implements a coarse-to-fine search of the target-type/target-pose search space. Tests at the higher levels in the tree are very general in that they test for the presence of target silhouette boundaries that could have been generated by any of a number of types of targets in a wide range of poses. Tests at the lower levels in the tree, in contrast, test for the presence of very specific target silhouette boundaries that could only have been generated by single types of targets in very limited poses. Figure 3 illustrates a sample decision tree. The tests are "relational," in that, rather than trying to recognize each target's silhouette independent of all other target silhouettes, the tests focus on aspects of the target silhouettes that differentiate the various targets.

The online target-recognition process applies the tests in the decision tree to each pixel of the image. The test associated with the root node of the decision tree is first applied to each pixel in the image. If a pixel passes this test, the tests associated with the node's "children" are applied to that pixel, and so on, until a terminal test has passed (meaning a target has been recognized at that pixel) or all tests fail (meaning that there is no target at that pixel). Because the center of a target can be located at any pixel in the image, this tree search is carried out at each pixel in the image.

Figure 3. A four-class
decision tree.*
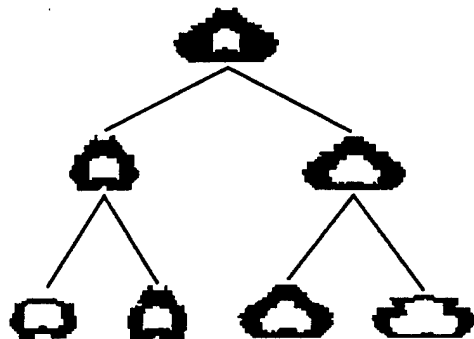
*To pass a node's test, a target's silhouette boundary
must lie in the dark region of the node's template.*

The only difference in the tests that are applied at the different nodes in the decision tree is that the target silhouette boundaries differ from node to node: the lower a node is in the decision tree, the more constrained the target-boundary test becomes. A target silhouette boundary exists at a pixel when the region around that pixel contains a sufficient number of edge points along the boundary of the associated target silhouette template. See [4] for more details.

4.2   Parallel ARTM Algorithm

As described in the previous section, the sequential ARTM algorithm carries out the same search algorithm at each pixel in the image. The algorithm is, therefore, inherently parallel. Our parallel implementation is as follows.

A copy of the image to be processed is first sent to each processor, and an assignment is made to each processor as to which pixels in the image it should examine. Each processor then independently applies the ARTM algorithm to its assigned set of pixels. When all processors have finished their tasks, the target regions found by each are merged into a single, consistent set of target regions.

In a system consisting of p processors, it is not possible to simply divide an image into p "blocks" (one for each processor) and expect a significant speedup of the algorithm, because only a few regions of the image contain targets or target-like clutter in a typical IR image. In the ARTM algorithm, much more computation is required in regions of the image containing targets and target-like clutter than in regions without targets. With this simple block-partitioning of the image, the few processors that receive image blocks containing targets will be busy, while the majority of the processors will quickly become idle.

To ensure good load balancing, it is essential to assign each processor roughly the same number of target (and target-like clutter) pixels. To this end, we assigned every pth column of pixels to the same processor, as illustrated in figure 4. When the number of processors in a system is significantly greater than the expected number of pixels across a target or

8

Figure 4. Assignment of pixels to processors.*

Processor number: 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
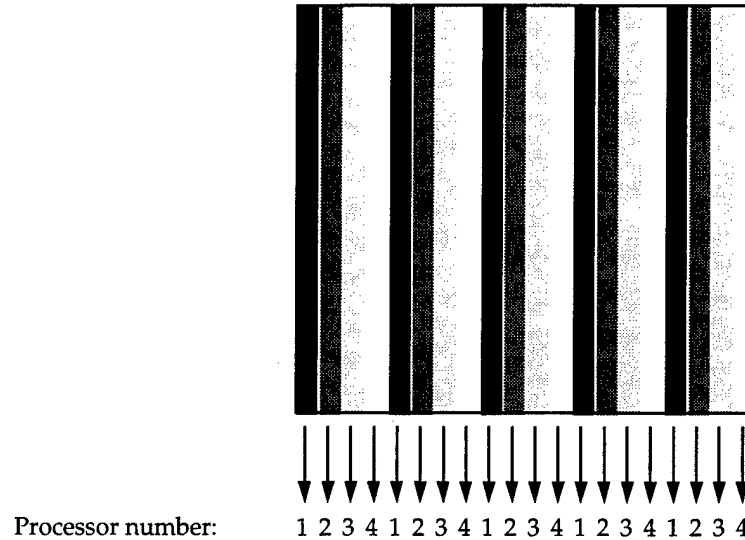
* In this example, p = 4.

target-like clutter, even this load-balancing scheme is not effective—some processors will receive no target pixels and will, therefore, spend much of their time idle. In such a case, it is easy to devise other schemes in which the pixels assigned to each processor are uniformly distributed over the image, which will uniformly distribute the target pixels to the p processors.

We demonstrated above that the ARTM algorithm is highly parallel. However, to obtain an efficient parallel solution, the processors must also have a fast mechanism to share program data (which, in our system, includes images, target silhouette templates, and target regions of interest). We experimented with two mechanisms for sharing data: shared memory and message passing. The performance of the system using each of these mechanisms is described in section 6. Since, in many processors, the use of shared memory is very limiting due to memory-contention problems, we concentrated our efforts on the message-passing architecture.

In our message-passing implementation, processors need only communicate during initialization (to obtain a copy of the image to be processed) and during the last stage of the algorithm (where the targets detected by each processor are merged into a single, consistent set of target detections). These communications can be achieved most efficiently when the processors are organized in the hierarchy shown in figure 5. This organization is a result of the physical organization of the quad-C40 board, described earlier: each board contains four processors, each with three internal communication ports (linked to processors on the same board), and three external communication ports (linked to processors on other boards). The quickest way to broadcast a message with this architecture is to use the following procedure.

9

Figure 5. A
four-level
processor
interconnection
network.*

Level

0

1

2

3
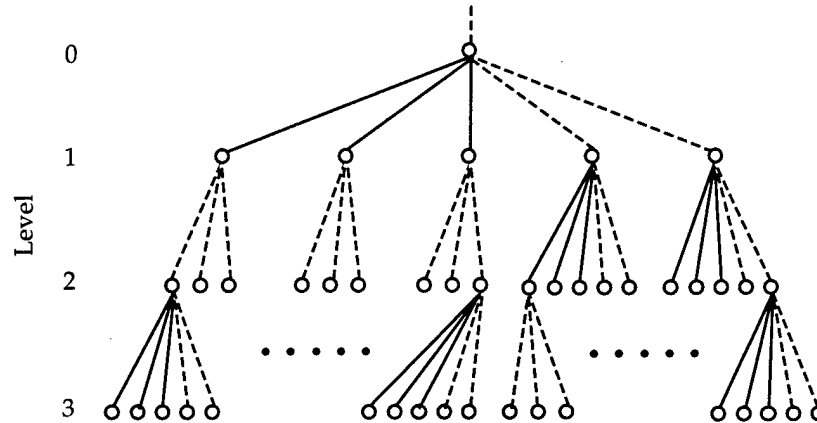
*Each circle represents a processor. Solid lines depict connections between internal
communication ports. Dotted lines depict connections between external communica-
tion ports. Images propagate from the top level to the bottom. Results propagate from
the bottom level to the top.*

One processor on each board receives messages from another board via an
external communication link. When the processor receives the message, it
sends it to the other three onboard processors (using the internal
communication links), and to processors on two different boards (using
the external communication links). The processors that are initialized via
the internal communication links then send the message out (using exter-
nal communication links) to three processors on three different boards.

5. Analysis

In this section, we analyze the scalability of the parallel ARTM algorithm
for the average case behavior. There are many performance metrics that
can be used to measure the scalability of a parallel system [6]. The system's
speedup as a function of problem size and number of processors is the
metric that we use here. Our problem size is given by n, where the size of
the image to be processed is $n \times n$ pixels. The speedup, S, of a parallel sys-
tem is defined as the ratio of the time required to run on one processor, T_1,
to the time required to run on p processors, T_p:

$$S = \frac{T_1}{T_p} . \tag{1}$$

The following analysis assumes that the parallel ARTM algorithm's load-
balancing scheme enables all processors to finish the decision tree searches
at the same time, so that the processors experience no idle time between
the end of the search and the start of target list merging. For this to occur,
each processor must be assigned roughly the same number of target/
clutter and nontarget/nonclutter pixels. This is possible when the number
of target/clutter pixels is much larger than the number of processors in the
system.

Our analysis of speedup is based on both the run-time behavior of an actual system and a high-level complexity analysis of the algorithm. We describe the algorithm in terms of a number of high-level, basic operations. The time required to execute each type of operation is determined by measuring the run time of the operation on our four-processor system. Using this timing data, we can generate an equation for the run-time complexity of the algorithm for any problem size and number of processors and, using this formula, we can calculate the system's speedup. The basic operations and their measured run times are as follows.

1. *Transfer an image between two processors that have a direct connection in the network.* Pixels are transferred at a rate of $c_1 = 1.6 \times 10^{-7}$ s per pixel. (This rate actually varies slightly with the image size, but we assume that it is constant.)

2. *Perform the decision tree search on a single pixel in the image.* The time to perform this operation can vary greatly from one pixel to the next, but for our average-case complexity analysis, we used the average value of $c_2 = 4.3 \times 10^{-3}$ s per pixel.

3. *Cluster pixels into target detections.* If we assume that the target and clutter rate per pixel is constant, the clustering rate per original image pixel will also be constant. We measured this rate to be approximately $c_3 = 6.1 \times 10^{-8}$ s per pixel.

4. *Transfer a list of target detections between two processors that have a direct connection in the network.* The size of this list depends on the number of targets and the amount of clutter in the image. Because the ARTM algorithm never detects more than a few targets or false alarms in an image, we can assume that the size of this list is essentially constant; therefore, any list of target detections can be transferred in constant time, $c_4 = 1.0 \times 10^{-3}$ s.

5. *Merge two lists of target detections into a single, consistent list of target detections.* Because we assume that the length of a list of target detections is constant, the time to combine two lists (c_5) will also be constant. We estimate that $c_5 = 3.0 \times 10^{-3}$ s.

Using the basic operations detailed above, we can calculate the run times of the sequential and parallel algorithms. The sequential algorithm consists of a tree search (time c_2) for each of the n^2 pixels, plus the pixel clustering (time c_3) for n^2 pixels. The sequential run time is, therefore,

$$T_1 = (c_2 + c_3)n^2. \tag{2}$$

If we let $L(p)$ denote the number of levels in the processor interconnection hierarchy containing p processors, for the parallel algorithm we have the following run times.

1. *Image propagation time*—Each processor may need to send the image to up to five lower-level processors in the interconnection network. The total image propagation time is, therefore, $5c_1n^2$ s for each of the $L(p) - 1$ propagation steps.

11

2. *Tree search time*—The tree search time is c_2 s for each n^2/p pixels.

3. *Pixel clustering time*—The pixel clustering time is c_3 s for each n^2/p pixels.

4. *Results propagation time*—A processor may need to receive target-detection lists from up to five lower-level processors in the interconnection network. Because a processor can read from only one of its communications channels at a time, results propagation is $5c_4$ s for each of the $L(p) - 1$ propagation steps.

5. *Target list merge time*—A processor can have up to six target lists that must be merged (counting its own). Since the lists are merged in pairs, five merges may be required; therefore, the merge time is $5c_5$ s for each of the $L(p) - 1$ merge steps.

The total parallel run time is then

$$T_p = (L(p) - 1) \times (5c_1 n^2 + 5c_4 + 5c_5) + (n^2/p) \times (c_2 + c_3) \ , \tag{3}$$

and the speedup as a function of n and p is

$$S = \frac{(c_2 + c_3)n^2}{\left(L(p) - 1\right) \times \left(5c_1 n^2 + 5c_4 + 5c_5\right) + \left(n^2/p\right) \times \left(c_2 + c_3\right)} \ . \tag{4}$$

To evaluate the equations above, we need an expression for $L(p)$, the number of levels in the interconnection network containing p processors. (For our purposes, we can assume that all levels in the interconnection network are full.) Let $r_d(k)$ denote the number of processors of degree d (i.e., that have d children) at level k in the network. From figure 3, we can see that

$$r_5(0) = 1 \ , \quad r_3(0) = 0 \ ,$$

$$\left. \begin{aligned} r_5(k) &= 2r_5(k - 1) + 3r_3(k - 1) \\ r_3(k) &= 3r_5(k - 1) \end{aligned} \right\} k \geq 1 \ . \tag{5}$$

Solving these linear recurrence relations, we get

$$\left. \begin{aligned} r_5(k) &= \beta_1 \alpha_1^k + \beta_2 \alpha_2^k \\ r_3(k) &= \beta_3 \alpha_1^{k-1} + \beta_4 \alpha_2^{k-1} \end{aligned} \right\} k \geq 1 \ , \tag{6}$$

where

$$\alpha_1 = 1 + \sqrt{10} \ , \quad \alpha_2 = 1 - \sqrt{10} \ ,$$

$$\beta_1 = \frac{\sqrt{10} + 1}{2\sqrt{10}} \ , \quad \beta_2 = \frac{\sqrt{10} - 1}{2\sqrt{10}} \ ,$$

$$\beta_3 = \frac{3\sqrt{10} + 3}{2\sqrt{10}} \ , \quad \beta_4 = \frac{3\sqrt{10} - 3}{2\sqrt{10}} \ . \tag{7}$$

12

The number of processors in level k of the network is then $r_3(k) + r_5(k)$, and $L(p)$ can be calculated by the expression

$$L(p) = min_l \left[\sum_{k=0}^{l-1} \left(r_3(k) + r_5(k) \right) \geq p \right] . \qquad (8)$$

Figure 6 shows the theoretical speedup of the parallel ARTM algorithm as a function of p for a number of values of n. The figure shows that larger speedups can be obtained as the problem size increases. For a given problem size, however, the overhead due to interprocessor communications limits the obtainable speedup when the number of processors is increased. Table 1 presents an example of how the system can be "scaled" to maintain an approximately fixed level of performance (i.e., constant run time) as the image size increases. The number of processors required is roughly proportional to the number of pixels in the image.

Figure 6. Speedup of the parallel ARTM algorithm as a function of the number of processors.

Image size

N = 16
N = 32
N = 64
N = 128
N = 256
N = 512

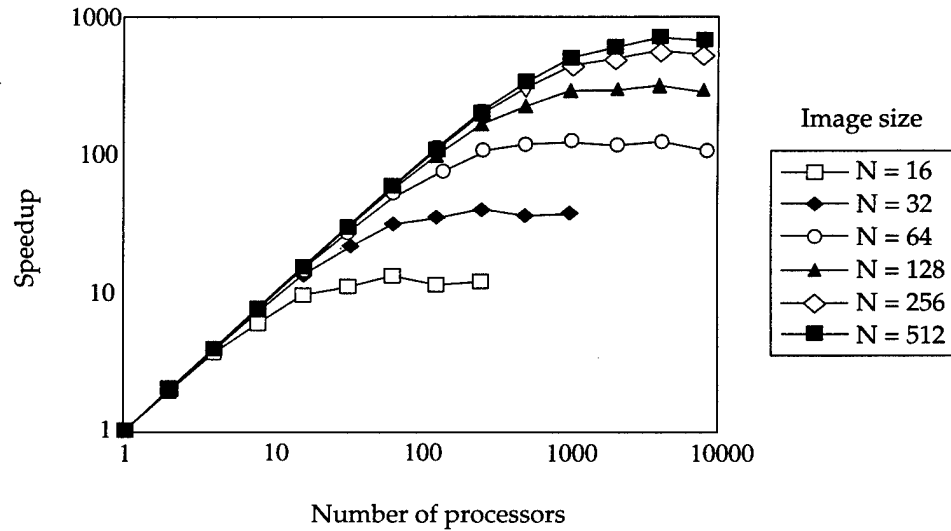Speedup

Number of processors

Table 1. Number of processors required to maintain a fixed performance level (< 1 s run time) as image size increases.

Image size	Run time (s)	No. of processors
8 × 8	0.275	1
16 × 16	0.572	2
32 × 32	0.902	5
64 × 64	0.975	19
128 × 128	0.991	79
256 × 256	0.999	390

6. Performance

Because it was easy to use, we chose shared memory for interprocessor communications in our first parallel implementation of ARTM. All shared data structures were stored in a single memory that was shared by all processors. Table 2 lists the elapsed run times and speedups for a number of test images when the algorithm is run on one to four processors. As is apparent from the table, contention for the shared memory becomes a problem when running with three and four processors, and severely limits the algorithm's speedup.

Our second implementation of ARTM performed message passing in a distributed memory architecture. After the first processor received the image data from the host, it broadcasted the data to the other processors in the system via its parallel communication ports. Table 3 lists the elapsed run times, the average speedups, and theoretical speedups for this system. Here, we obtained a nearly linear speedup of the algorithm. Figure 7 compares the speedups of the shared and distributed memory architectures.

Table 2. Elapsed time (in seconds) and speedup for the shared memory algorithm. (Shown as elapsed time/speedup.)

| Image | Number of processors | | | |
	1	2	3	4
metd0900	78.2/1	42.3/1.8	34.7/2.3	32.7/2.4
metd0909	107.5/1	56.6/1.9	46.4/2.3	42.7/2.5
metd1268	133.3/1	70.7/1.9	59.1/2.3	54.5/2.4
metd2523	121.0/1	63.9/1.9	52.2/2.3	48.8/2.5
metd2656	116.2/1	61.6/1.9	51.7/2.2	47.5/2.4
metd3241	83.8/1	44.8/1.9	37.7/2.2	34.6/2.4
metd3321	84.7/1	45.3/1.9	37.1/2.3	34.5/2.5
metd3785	97.1/1	51.7/1.9	44.0/2.2	40.3/2.4
metd4446	124.0/1	65.7/1.9	55.7/2.2	50.9/2.4
metd5045	79.7/1	42.5/1.9	35.3/2.3	33.1/2.4
metd5264	88.6/1	48.3/1.8	40.1/2.2	38.0/2.3
Average speedup	1.0	1.9	2.2	2.4

Table 3. Elapsed time (in seconds) and speedup for the distributed memory algorithm.

| Image | Number of processors | | | |
	1	2	3	4
metd0900	59.4/1	29.9/2.0	20.3/2.9	15.3/3.9
metd0909	75.6/1	37.8/2.0	25.7/2.9	19.3/3.9
metd1268	91.1/1	45.6/2.0	30.6/3.0	23.0/4.0
metd2523	84.9/1	42.3/2.0	28.3/3.0	21.4/4.0
metd2656	79.6/1	39.7/2.0	26.8/3.0	20.3/3.9
metd3241	58.1/1	29.3/2.0	19.7/2.9	14.9/3.9
metd3321	59.5/1	30.0/2.0	20.2/2.9	15.3/3.9
metd3785	66.4/1	33.4/2.0	22.6/2.9	16.8/4.0
metd4446	84.3/1	42.0/2.0	28.4/3.0	21.3/4.0
metd5045	55.4/1	27.8/2.0	18.9/2.9	14.4/3.8
metd5264	66.3/1	33.4/2.0	22.7/2.9	17.2/3.9
Average speedup	1.0	2.0	2.9	3.9
Theoretical speedup	1.0	1.998	2.996	3.993

Figure 7. Speedup of distributed and shared memory algorithms.

Number of processors
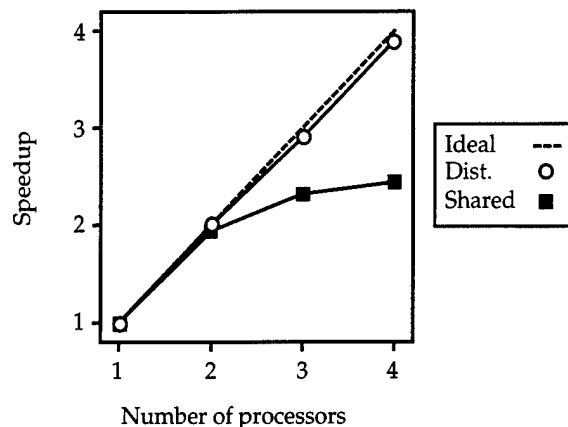
7. Conclusions

For this study, we parallelized the ARTM ATR algorithm and implemented it on a scalable architecture of C40 processors. We demonstrated the scalability of the system empirically for a small number of processors, and theoretically for larger numbers of processors. Using these results, an estimate of the number of processors required to obtain a given level of performance in a particular application was derived.

The commercial, off-the-shelf hardware used in this system and the open-architecture nature of the hardware and software makes the system affordable and easy to work with. As such, it is appealing to look at other applications that might benefit from the use of scalable architectures. We are currently considering the feasibility of implementing a multiple-hypothesis tracker [7] on this system. There are several broad battlefield applications for which a scalable architecture approach might also be feasible, including:

1. Terrestrial, atmospheric, and space-borne sensor images merged with digital map and entity (friendly and enemy) data to provide realistic fly-through simulations directly to the battlefield prior to tactical engagement.

2. The deployment of coordinated, autonomous air and ground robotics providing services such as real-time reconnaissance, surveillance, and target acquisition; decoy and mine detection and clearance; and electronic warfare.

3. The fusion of all available sensor information for real-time situation assessment and awareness, combined with advanced human-computer interfaces (e.g., visualization, natural language, intelligent database access) to enable rapid assimilation of this critical information.

4. A mobile, distributed command and control network that supports real-time, worldwide teleoperations. (A simple example of this network is the telemedical application with which Conus medical experts are interactively supporting medics in field operations).

5. Adaptive, hybrid, terrestrial-satellite communications networks to support items 1 through 4.

15

Acknowledgment

References

1. K. Hwang and F. Brigg, *Computer Architecture and Parallel Processing,* McGraw-Hill, Inc. (1984), 121–123.

2. *Considerations in Choosing a High-Performance Floating-Point DSP,* technical white paper by Texas Instruments, Inc. (1993), 7.

3. *SPOX-LINK Programming Guide,* Spectrum Microsystems, Inc. (1993), 11–13.

4. A. Kramer, D. Perschbacher, R. Johnston, and T. Kipp, *Relational Template Matching for FLIR Automatic Target Recognition,* Proc. SPIE, Vol. 1957 (1993).

5. S. Savitt and R. Suresh, *ATR Relational Template Matching,* report prepared for the Army Research Laboratory by Alliant Techsystems, Inc., and Mathematical Technologies, Inc., under contract DAAB07-90-C-F427 (September 1993).

6. A. Grama, A. Gupta, and V. Kumar, *Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures,* IEEE Para. Distrib. Tech. (August 1993).

7. D. B. Reid, *An Algorithm for Tracking Multiple Targets,* IEEE Trans. Auto. Cont. (December 1979), 843–854.

Distribution

Admnstr
Defns Techl Info Ctr
Attn DTIC-OCP
8725 John J Kingman Rd Ste 0944
FT Belvoir VA 22060-6218

Hdqtrs Dept of the Army
Attn DAMO-FDQ MAJ M McGonagle
400 Army Pentagon
Washington DC 20310-0460

US Army Artificial Intelligence Ctr
Attn SAIS-AI
107 Army Pentagon
Washington DC 20310-0107

US Army Rsrch Lab
Attn AMSRL-IS B Broome
Attn AMSRL-WT-WC C Shoemaker
Attn AMSRL-WT-WC G Haas
Attn AMSRL-WT-WC J Bornsetin
Attn AMSRL-WT-WC T Huag
Aberdeen Proving Ground MD 21005-5067

US Army Rsrch Lab
Attn AMSRL-IS D Hillis
Attn AMSRL-IS L Tokarcik
Attn AMSRL-IS P Emmerman
Attn AMSRL-IS-PA M Salonish
Attn AMSRL-IS-PA S Ho (6 copies)
Attn AMSRL-IS-SA J DeHart
Attn AMSRL-IS-SA L Sadler
Attn AMSRL-IS-SA P David (12 copies)
Attn AMSRL-IS-SA S Balikirsky
Attn AMSRL-IS-SA T Mills
Attn AMSRL-OP-SD-TA Mail & Records
  Mgmt
Attn AMSRL-OP-SD-Tl Tech Lib (3 copies)
Attn AMSRL-OP-SD-TP Tech Pub (5 copies)
Attn AMSRL-SE-RS T Kipp
Attn AMSRL-SE-RT M Hamilton
Adelphi, MD 20783-1197